THE ACE CHALLENGE

The past, the present, and the future of games technology...

THE ACE CHALLENGE

This booklet will not only explain to you the promise – and the problems – of producing a world-beating games micro. It will also give you our technical expert's specification of a games-playing machine for the '90s, and news of a £20,000 challenge to the computer games industry. Here's the line-up...

BIG BROTHER IS WATCHING...

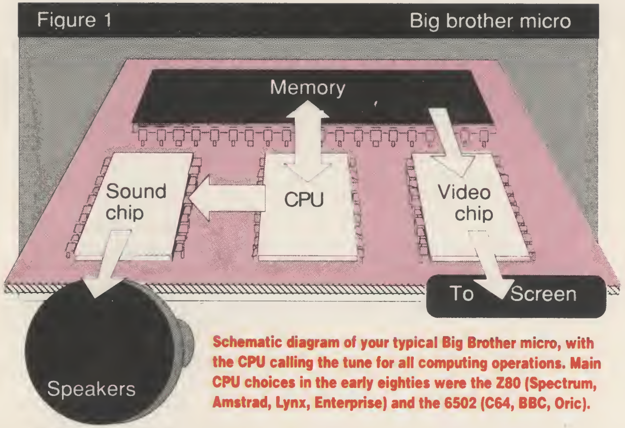
Over the last five years, there's been a revolution in the design of entertainment micros. Before we can look forward, however, we need to look back. We'll start with an examination of the original home computer designs and then compare them with today's micros.

Back in the early mists of time – say around 1984 – the dominant home computer designs could be called, appropriately enough given the year, 'Big Brother' computers, due to their heavy reliance on one, all-powerful (or not so powerful) central processing chip (CPU). Anything that gets done in your Spectrum or Vic20 gets done by the CPU – you want a sprite displayed on screen? The CPU copies the image data into screen memory. You want to play a tune on the speaker? The CPU sends it to the sound chip, one note at a time. You want to perform some complex calculation to do with 3D geometry? The CPU crunches those numbers for you.

Since the CPU does (just about) everything in this kind of set-up, its power is the single most important factor in the micro's performance. The only way to increase its power is either by 'clocking' it faster – making it do more operations every second – or increasing its 'data width' – that is, making it do more in each of those operations. Of these two methods, increasing the data width was the one that looked most promising back in 1984. There were 16- and even 32-bit chips in the offing to replace the sluggish 8-bit ones that got the computer boom started.

FASTER, FASTER!

You can think of speed and data width in terms of road planning. If you want a road to carry more traffic you could increase the speed-limit, but you can't keep doing this indefinitely: once you get much above 70mph things get difficult, and at 180mph you won't have a working system at all.



Schematic diagram of your typical Big Brother micro, with the CPU calling the tune for all computing operations. Main CPU choices in the early eighties were the Z80 (Spectrum, Amstrad, Lynx, Enterprise) and the 6502 (C64, BBC, Oric).

As with cars, so with chips. There's a theoretical limit to how fast a processor can run – data simply cannot move faster than the speed of light – but far more importantly, there's a great big practical one. A processor can only run as fast as the memory it has to work with. Every time a processor wants an instruction, it has to read a memory location. If the instruction tells it to read or write other locations – and in a game, such instructions take up a lot of the processor's time – that's more memory that'll have to be accessed.

If you buy a fast PC AT clone, its 80286 CPU might well be running at 16MHz or more and (at this speed) accessing memory 8 million times a second. Speeds this high cause problems (see panel on page 6) so the usual way round this is to use a "cache" – that is, a temporary storage area – of fast static random access memory (SRAM). The SRAM stores the 16K or so of memory locations that the CPU needs to access most often, so there's far less need to look at the slower DRAM.

This cache method, however, causes problems where games are concerned. A PC program may well spend 90% or more of the time accessing the same few locations, but a modern game typically needs to get at 100K or more on a regular basis. For starters there's the front screen (the one you're looking at) and the back screen (the one the game's getting ready to show you) at 30K or so each. Then there are the sprites, sprite masks, sound data, and the tables that describe how objects behave. On top of that there's the actual program code itself: in games this tends to be unusually long-winded, because such code often runs faster. (A classic Amstrad CPC technique uses 80 identical instructions in a row to perform just part of one task, and there are parallels on most machines.)

The only real way out of the memory bottleneck is to fit a machine with SRAM chips throughout, but at modern prices that'd be incredibly expensive. With luck we should see prices start tumbling around 1992 or 1993, as manufacturing techniques improve. Even so, that'll only just get cheap memory up to the sort of speed that processor chips can run at now.

MEMORIES ARE MADE OF THIS...

At the moment there's only one kind of memory cheap enough to use in large quantities. This is dynamic random access memory (DRAM) and it isn't terribly fast stuff. You can only read or write to a DRAM chip about 5 million times a second and still get reliable results. This may sound like it'd be fast enough for anybody, but in practice that's not so: an 8MHz 68000 like the one in an Atari ST expects to get at its memory 2 million times a second, so clearly the limit isn't that generous. Doubling the chip's speed might be okay, but beyond that you're going to have problems. It's no good having a processor that wants 8 million accesses a second if the memory can't keep up. (NB this "memory access" rate isn't the same thing as a chip's clock speed. Most processors only access memory one "tick" in every two or, like the 68000, one in every four.)

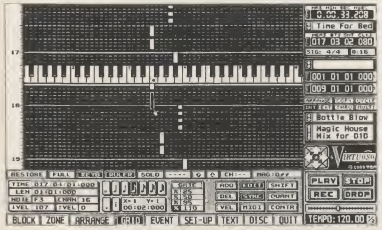
FEEL THE WIDTH

Increased speed isn't the only way to get a chip doing more work. You can also try increasing the data width. Let's go back to that traffic analogy for a moment to see how this works.

If a road can't handle the weight of traffic required and you've already raised the speed limit as high as you dare, a good alternative is to add extra lanes, turning the road into a dual carriageway or even a motorway. The cars may not actually move faster in this kind of set-up, but you can fit more of them into every mile of road so the thing actually carries more traffic.

Just as you can't keep boosting a system's speed indefinitely, so you can't carry on increasing the width for ever. For one thing, beyond a certain point that extra width isn't actually very useful. A 100-lane highway is a difficult, expensive thing to build, and doesn't do anyone a great deal of good anyway. The chances are, you'll never get 100 cars side by side, so what's the point of all that extra expense and effort?

Again, the same goes for processors. Home computer processors were originally only 8 bits "wide" – that is, they could only handle data 8 bits at a time. Now, 8 bits isn't actually a great deal of data. It's not enough to store the internal pointers every program needs; it can't hold the size of co-ordinate a typical game has to work with nowadays; nor can it describe



MIDI is a standard of communication between computers and musical instruments, and all the codes in the MIDI standard are 8-bit (i.e. binary numbers with eight digits, equivalent to the range 0-255 in decimal). For this reason, as with chess (see p.9) MIDI programs could show a degree of 'wastage' when running on a 16-bit system.

even one row of a modern sprite. Because of this, 8-bit chips had to deal with these quantities piecemeal, taking several memory accesses to read or write each one. It came as a great relief then when 16-bit chips like the 8086 (modern PC clones) and the 68000 (Atari ST, Commodore Amiga) started appearing on the market.

WASTED BITS?

Because so many quantities in the games world were already at least 16 bits wide, these 16-bit processors really could get through data at twice the rate on most tasks – and that's before you've even mentioned the increased clock speeds they can manage. It's worth noting that "most tasks" qualification here though: some kinds of game don't rely on 16-bit quantities very heavily, and so get relatively little benefit from increased data width. A classic example is computer chess, which has a "natural" width of just 8 bits (because of the shape of a chess board). Chess programs spend so much time dealing with 8-bit quantities that the extra width is largely wasted.

Such examples of wastage are rare on 16-bit machines, but become a great deal more common once you step things up to 32 bits. Take the Acorn Archimedes for example: its ARM chip can read or write numbers up to 4 billion in just one memory access. Though there are important uses for this kind of width – especially in copying

FEEL THE WIDTH...

Increasing a processor's data width isn't cheap. If you double the number of bits it can handle, you double the number of metal "legs" it needs and the number of copper "lines" you'll have to etch onto its system board. At the moment, 32-bit processors are the widest things you can economically manufacture and make boards for. Though cheap techniques could doubtless be developed for 64- or 128-bit chips, it's hard to see why anyone would bother. We're already getting diminishing returns on the step up from 16 to 32 bits, and going further would be still less beneficial.

sprites or "socking in" areas of a solid 3D display – there are equally going to be a heck of a lot of cases where those extra bits just aren't necessary.

A NEW APPROACH

What's needed is a brand new approach to the problems of getting a powerful machine – and that's just what we're starting to see now. You see, the real problem with the classic 1984-style micro is that everything's so centralised. It's as if everything that happened in Britain had to be done in London. You want to work? You want to eat, watch a film or go shopping? Go to London. Looked at that way, the next time you get stuck in a traffic jam in the capital you might like to consider that what's going wrong isn't really to do with traffic at all. The problem is that one city is trying to cater for the needs of 56 million people, and that isn't really going to work. In computing, as in real life, what we need is a system with more 'cities' – more places where work gets done and things happen. Instead of one Big Brother chip running the whole show, we need to spread the effort around a little. And that's where the modern machine comes in...

THE COCKTAIL MACHINE

Despite their apparent power, machines like the ST and the Archie are still based very much on the old "Big Brother" idea of one central processor doing everything, but they're a dying breed. The ST holds its own because of the power of the 68000 processor and the current state of games programming practise, which is still recovering from the limitations of 8-bit technology. The Archie uses

RISC technology (see next section, and ACE Issue 26 pp109-111) to drive the Big Brother concept to the limit with an ultra-powerful CPU, but the machines coming to the fore now feature less powerful CPU's and farm out heavyweight tasks to specialised processors.

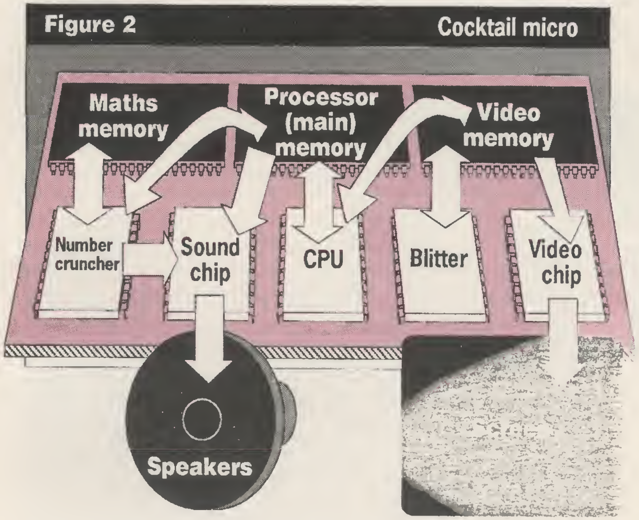
The modern games machine is a big step forward in several ways, particularly in the three separate chips (CPU, blitter and number-cruncher) that do the processing work. The blitter and cruncher specialise in bread-and-butter games tasks that the CPU performs slowly. Because they're specialised they can get the job done much faster – and because of the way memory's divided, they can do this while the CPU's doing something else entirely.

Note also how the sound chip finally gets what the video chip had all along: the ability to read its own data direct from memory, rather than being spoon-fed by the CPU. Though there isn't a special "sound memory" area, the sound chip can read tunes and waveforms set up for it beforehand just as the video chip displays pre-arranged pictures. This means music can run independently of onscreen action, and quite complex sound effects can be triggered from a single CPU instruction.

The blitter's an increasingly common sight on circuit boards and is also, thanks to Commodore's Amiga publicity, a household word. The name itself is a contraction of Block Image Transferrer, and that's exactly what the chip's for – transferring block images, rectangles of picture data, from one place to another. Your average blitter can trim away unwanted borders so as to transfer non-rectangular images, and can also turn its hand to simple art task like filling rectangles with solid colour or drawing straight lines. Any CPU can do this kind of thing itself, but the blitter has a big advantage – it's much faster.

HORSES FOR COURSES

The fact is, most CPU's are designed by digital electronics boffins or their computer science equivalents, not by games programmers. It's hardly surprising then that a typical CPU is very quick at changing stack frames (not much use in a shoot-em-up) or switching between



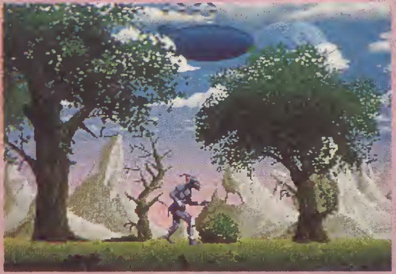
As far as current home micros are concerned, the Amiga represents a typical early 'cocktail' approach with its PAD (three specialised chips nicknamed Paula, Agnes, and Denise) which oversees memory and video operations. Here's a diagrammatic representation of a typical 'cocktail' micro – notice how the diagram is now far more complicated, with two brand-new chips adding their processing muscle to the system.

processes (ditto) but pretty sluggish at shunting pixels around. Blitters win out because pixel-pushing is all they do, so they can be as specialised as necessary.

Number-crunchers win out in their chosen field for precisely the same reason. These dedicated maths chips have actually been around rather longer than blitters – the Intel 8087 maths chip has been boosting the PC's arithmetical prowess almost since the start of the decade – but for years they've been used almost exclusively by business and scientific software. That's all set to change now, thanks to Flare's innovative Konix console design. The Konix's Digital Signal Processor Chip (DSP for short) is in fact a highly streamlined, very powerful arithmetic engine. Specialising in maths as it does, it can burn through 3D geometry or fractal generation faster than any modern CPU could hope to. It's odds on that, like the Amiga's blitter, this little belter of a chip will spawn a whole load of imitators over the next few years.

BIG BEASTIES

Psygnosis were the first UK company to release products on the ST, but interestingly since then they've moved over to development on the Amiga. Nowadays, they concentrate almost exclusively on the Commodore machine and release only converted or commissioned material on the ST. The reason for this is the 'cocktail' nature of the Amiga which gives them greater graphical and sound control over their products. Here's a screenshot from their game *Shadow of the Beast* which features several processor-intensive graphics and sound operations, including 13 levels of parallax scrolling in the outside scenes (as seen here) running at 50 frames a second, while a sampled sound track plays in the background, and large sprites are manipulated on-screen. Although the ST and the Amiga share the same 68000 CPU, the Atari's chip would be completely overwhelmed if it attempted this task, whereas the Amiga's 68000 can rely on the machine's custom chips for support.



DIY SILICON

The Konix Multi-System: in this modern 'cocktail machine' the CPU was needed only for housekeeping. Originally a Z80 was specified, but for marketing purposes it was decided to go for a 16-bit 8086. The moral is that in a cocktail machine, choice of CPU is not of paramount importance.

The great thing about chips like the Amiga's blitter or the Konix console's DSP is that they're custom-made. The boffins in a modern design team aren't tied to the processing power available "off the shelf", but can actually create their own power chips to fill particular needs. With something like a blitter this is actually quite straightforward, because of the simple, brute-force tasks the finished article has to handle. Number-crunchers aren't quite such a push-over, but even entire custom-made CPU's are perfectly possible. The only big expense involved is now the designer's time: electron-beam techniques mean that a prototype chip can cost just a few thousand pounds, and that's very cheap indeed.

Over the next few years the big task for micro designers will be to produce better custom chips, tailored more closely to games. To do this they'll have to pay far more attention to what programmers are saying. The pressing need at the moment is for a better blitter: the Amiga's takes too long to start up – it's more trouble than it's worth for small sprites – and, like the Konix blitter, is very little use for modern 3D games. Programmers are crying out for a chip that can draw the shapes used in games like Carrier Command and Virus. Triangles would do – more complex shapes can be built out of them – but as yet there's no sign of such a "tritter" chip being developed. What'd really work well would be a high-speed tritter combined with a purpose-built geometric number-cruncher. Rotating and scaling co-ordinates for perspective 3D games is a time-consuming business and, more to the point, a fearsomely difficult one to program. The sad thing is that this 3D trigonometry is universal – all such games have to perform the same calculations – so it'd be perfectly possible to design one "trigger" chip that everyone could use. Come on guys, how about it?

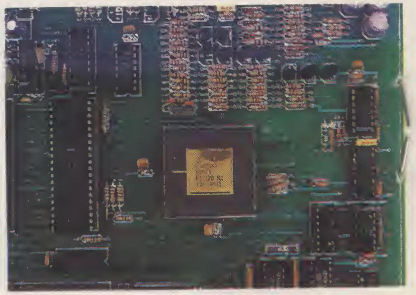
SEPARATION OF POWERS

Far more important than the simple existence of these new chips is the way they're linked in to the machine's memory. By giving the three processors of the system separate areas of memory to work on, they can actually run simultaneously most of the time. That is, the number-cruncher can work out a new set of co-ordinates while the blitter clears a fresh back-screen to work on and the CPU checks for object collisions. Not only are the blitter and number-cruncher doing their jobs faster than the CPU could, but they're actually doing them without interrupting the CPU's other tasks.

The idea of divided memory in fact predates both number-crunchers and blitters by several years. It was pioneered on the Sinclair Spectrum as a way of reducing "contesting" problems. In modern multiprocessor designs, contesting is the effect where two chips – the CPU and the blitter, say – both want to access the same area of memory at once, so that the less important one has to be temporarily frozen – or "locked out", as the technical term has it.

The Sinclair solution on the Spectrum was to split memory, so that the video chip could only access the first 16K (where the screen lived). The CPU had unqualified control over the remaining 32K, so that programs located there could run much faster. This meant a) that almost all Spectrum games were written to run in the upper 32K of memory, and b) that the 16K Spectrum was almost useless as a result.

As a bizarre footnote, the Amiga was actually designed to have a very similar division in its memory layout, but most of the Amigas in the UK today can't actually get the benefit of it. The Amiga design revolves around the idea of "chip" memory and "non-chip" memory: the video chip and blitter can only access chip memory, so programs located in non-chip memory can run at full speed even when the blitter's busy. Unfortunately, chip memory is defined as being the first 512K of memory in the machine, so unexpanded A500's don't actually have any non-chip memory. Look on the bright side though – since hardly any games use the blitter, the CPU usually has chip memory to itself anyway. (Why doesn't this sound like much of a bright side?)



The Konix ASIC (Application Specific Integrated Circuit) is typical of the large-scale integration of today's custom chips, containing sound and graphics handling, disk controller, blitter, ROM, fast RAM, arithmetic and logic unit, and control ports. This monster provides much of the power behind this state-of-the-art games machine...

BLANK MOMENTS

The Spectrum didn't have a blitter of course, but its CPU still lost valuable time waiting while the video chip accessed memory. As in all normal micros, the Spectrum's video chip had to read screen memory in order to know what it should be displaying: this kind of memory access has to take priority over anything the CPU's doing, because the video chip's work is time critical. If the video chip stops for anything, the picture will be disrupted. On the earlier ZX80 and ZX81, video accesses did not necessarily have priority: sure enough, when the CPU got busy, the picture collapsed!

THAT DOES NOT TRANSPUTE

Once you start dividing memory between different processing chips this way, suddenly the sky's the limit: you can create a micro as powerful as you need, with only cost to limit you. If one number-cruncher can't handle that geometry, why not get two on the job? If one blitter is too slow for all that shape generation why not get several of them on the job?

Getting a whole load of processors working at a single task this way is called parallel processing. In serious computing circles, parallel processing has been big news for quite a while now. The centre of attention here is the Inmos Transputer, a chip family specifically designed for parallel use. Last year Atari unveiled their ATW Transputer Workstation (nee ABAQ), based on the top-of-the-line T800 Transputer. The T800 is an odd chip in many ways: it has a cache of fast memory built in, can handle its own number- crunching, and is fiendishly expensive to manufacture.

If the price of the T800 can drop far enough over the next five years, it could well be the basis of some terrific mid '90s games machines. Unfortunately though, its current high price has a lot to do with its extreme complexity. Those built-in goodies don't come cheap and, arguably, could be handled far better off-chip.

At the moment though, if you want a chip that's specially designed to work in groups, the Transputer family is the only game in town. That is, unless some enterprising team wants to design a new parallel-minded processor of its own. Designing your own processor is no mean feat, but it can pay dividends: Acorn's in-house boffins came up with the ARM to power their Archimedes range, and the ARM is an impressive chip by anyone's standards. Thanks to the advent of RISC (Reduced Instruction Set Computer) technology, you can produce a state-of-the-art processor with really very little silicon.

HEARTS AND MINDS

Of course, turning out your own CPU does have one big drawback: no-one will know how to program it. When hit machines like the Spectrum and C64 came onto the market, they used central processors that were well known and already had plenty of fans. Even the ST and Amiga had the ground broken for them by Sinclair's ill-fated QL. A games machine with its own custom processor(s) is a very different matter. Why should a programmer bother to learn a brand new chip if the chances are no other manufacturer will ever use it? A processor with no past and little future looks like a bad thing to spend time on.

Acorn's Archimedes range uses their custom-designed ARM CPU, a very popular chip with programmers due to its simple instruction set and high speed. Unfortunately, the price of the machine denies it a mass market...

There's a flip-side to this though: if a designer's in touch with what programmers want, he can create a chip that they won't be able to resist. Programmers love well-designed hardware, and will go out of their way to work on it. If Acorn had built their ARM processor into an ST-priced games machine a year or two back, they'd have had an incredible rush of would-be ARM experts on their hands. Why? Because the chip is, by design, both easy and fun to program. The learning effort involved is very small, simply because the chip has been put together thoughtfully. By comparison the Konix console's 8086 processor is a complete brute to learn, and ugly even once you've mastered it. The Acorn route's a gamble, sure enough, but the right chip in the right machine stands every chance of winning the programmers' vote.

BRAINS ELSEWHERE

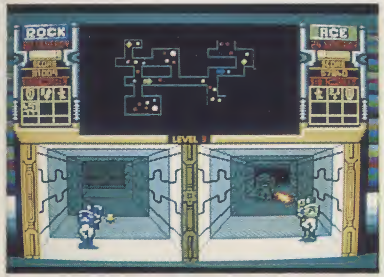
So far all we've really discussed is processing power – the rate at which a games machine can alter areas of memory, or perform calculations. While big-league processing power is always a handy thing for a machine to have, it's not the sole – or even an essential – qualification for a good games machine. Clever video hardware can give you terrific screen effects even with quite a puny CPU.

Take the Amstrad CPC and the Commodore 64 for example. In processing terms the CPC wins hands down – its 4MHz Z80A CPU has roughly twice the power of the C64's 1MHz 6510 – and yet, for mainstream games, the C64 is a much better machine. Why? It's not down to palette size or screen resolution – the CPC wins out on both counts – but rather is to do with the clever way the C64's hardware works out what to display on screen. The C64 has a whole bag of tricks between its memory and your TV, while the CPC's video chip is a straightforward, off-the-shelf piece of work.

Like most 1984-style machines, the CPC has a "memory-mapped" screen: a large slice of memory is used to describe what the screen display should look like at any one time, each little chunk of memory describing what colour one pixel (ie one dot of light) should be. Fifty times a second the CPC's video chip scans through this slice of memory, translating each pixel-description into the signals needed to display that pixel onscreen. If the appropriate chunk of memory says "make the next pixel light blue", the video chip simply translates this into monitor signals meaning "make the next pixel light blue". That's all it does: read a description, translate it into monitor signals, and then send those out.

In a memory-mapped system like this, a program changes the picture onscreen by changing the pixel descriptions stored in that slice of screen memory. If it needs to scroll the screen sideways one pixel at a time, it has to move all the pixel descriptions along one – and keep this up fifty times a second, to match speeds with the video chip's scanning process. For a full size screen, this is impossible: the CPC's Z80 just can't work that fast. If you wonder why horizontal scrollers are so poor on the CPC, there's your answer. (Even the ST, with all the power of its 68000, has problems in this department: again, direct memory mapping is the culprit.)

The C64 doesn't have this scrolling problem, because its screen display is much more loosely related to the contents of its screen memory-map. If you want the whole screen scrolled one pixel sideways, you don't have to move those pixel descriptions in memory at all: you can simply instruct the video chip that you want the picture moved sideways, and it'll take this into account when it's translating the descriptions into signals. Giving the video chip instructions like this takes the C64 only a few millionths of a second – around a ten-thousandth of the time that the CPC's brute force method needs.



The C64's hardware design led to the programming of many 'sprite-based' games in the early '80s.

The same is true, to a lesser extent, with sprites. On a directly memory-mapped machine, animating a sprite involves scrubbing its old image out of screen memory and copying a new one in. This isn't necessary on the C64: here you just send the video hardware an instruction telling it to move an existing sprite, or swap it for a new one. The sprite images are never part of the memory map, but rather are drawn "over the top of" it, by the video hardware, during the translation process. Again, what would be time-consuming on a directly mapped machine takes just a few millionths of a second on the C64, thanks to its video hardware.

SPRITES IN THE '90S?

Here's a conundrum for you: like the 64, the Commodore Amiga has video hardware that provides sprites and pixel-at-a-time scrolling. It also has a few other video tricks that could come in handy for games. Almost none of this stuff gets used in commercial games though: with a few exceptions, they all get by with the Amiga's 68000 central processor doing all the work. Why should the features that sold the C64 be ignored on the Amiga?

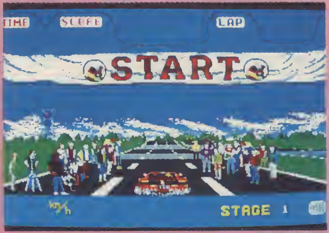
The stock answer here is that most Amiga games are ported straight over from the ST, and ST games have to do everything with the 68000 (not having any clever video chips to help out), so the Amiga versions work that way too. This is only half the answer. If programmers just port ST games straight over to the Amiga, it's because they can get away with it – and that in turn is because the Amiga never really needed all that clever video hardware in the first place.

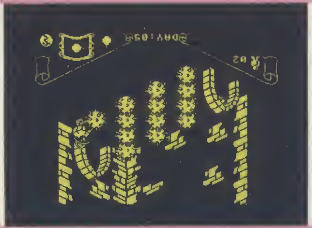
Whenever you see a games machine with fancy sprite or scrolling hardware, you really have to ask yourself "Why is this necessary?". With a C64, the answer is obvious: that 1MHz 6502 wouldn't have the power for a decent game. Without its extra hardware the C64 would, like its predecessor the VIC 20, be an underpowered micro with little to recommend it. When you look at the Amiga though, things aren't quite so straightforward. In the sprite department at any rate the 68000 is easily up to the task itself, and doesn't present the programmer with any new learning tasks or technical restrictions.

There's a more general point about video hardware that the Amiga underlines here: it's very limiting. Video hardware that's designed for pixel-scrolling, eight-sprite games won't be any help for a *Knight Lore*-style isometric 3D adventure. It won't do anything to speed up solid 3D graphics either, or to handle those vast chunks of landscape and foliage used in *OutRun*-style head-on games. In other words, for all that the Amiga spearheaded the Cocktail Micro approach, its video hardware was designed for a style of game that has all but died out. For a modern coin-op conversion, the Amiga has to rely on its 68000 and blitter. As we've already seen, the blitter's not exactly futureproof – it's very little use for solid 3D – so a few years down the line the Amiga really will, for graphics purposes, be on a par with the ST. It'll still have bags of extra hardware of course, but nothing worth using on up-to-date games.

CHANGING THE RULES...

As gamestyles develop, so do the demands on hardware. This isometric 3D Ultimate game was originally developed on the Spectrum. The screen backgrounds simply flipped into place when you moved rooms. OutRun, on the other hand, really needs either a fast processor or a blitter to shift the background as you move round the track.





Above OutRun

Below Knight Lore

NEW AGE VIDEO

If blitters and number-crunchers need to become "tritters" and "triggers", what should sprite-and-scroll video chips become? For starters, the idea of pixel-scrolling is going to pretty much disappear.

The pixel scroll was an important idea in the first place because of the way screens were memory-mapped. In the old days, on machines like the CGA-adaptor PCs, Spectrums and the like, each byte of memory would represent several pixels on screen. This meant that moving the memory map a byte at a time – the smallest movement you could comfortably make – would shift the picture more than one pixel at a time. There's a similar problem with the memory map favoured on EGA-adaptor PCs, the ST and the Amiga: a one-byte movement in memory causes an eight-pixel movement onscreen.

This style of screen memory-map is starting to look old-fashioned now, and the '90s should see it die out altogether. Far better is the byte-per-pixel style used by the Konix console design and the Acorn Archimedes. Here a one-byte movement in memory equals a one-pixel movement on screen, so it's quite possible to scroll the screen one pixel at a time without video help. Designers might as well build in a hardware scroll feature anyway, but this will no longer be a special selling point: with a byte-per-pixel screen, hardware scrolling will automatically be pixel-smooth.

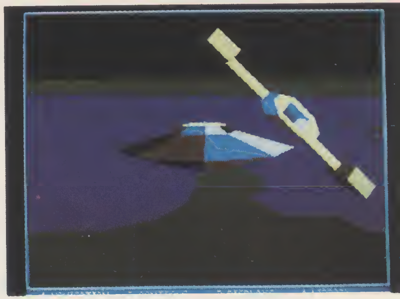
At the same time as this, we can expect to see a new kind of sprite showing up on home machines. rather than being a fixed size and shape, the super-sprite of the nineties will be completely flexible. The video chip will not only draw it onto the backdrop for you but also enlarge, reduce, rotate or distort it as needed. As objects approach the player they'll expand, to fill the screen if necessary, just by virtue of simple instructions to the video chip, without any need for the programmer to store different-sized versions.

Super-sprites like these are already commonplace in the arcades – Sega games like *ThunderBlade* are full of them – but a home equivalent has yet to surface.

The biggest challenge for tomorrow's video hardware goes a lot deeper than scrolling or sprites however, and could completely change the look of games. It centres round display technology, and the way computers have used it up till now.

For games, the basic home display has usually been a colour television. Modern colour TVs don't have a very high resolution, and typically can only handle 320x200 pixels comfortably. During the '90s we can expect to see a new TV standard making inroads in the UK, and bringing a rather higher resolution picture with it. Even if the jump is large – to 800x600 pixels, say – this won't catch up with accelerating computer power. Already the Atari Transputer Workstation can handle 16-colour million-pixel screens at game speeds, and that's before the heavy-duty games programmers have been at it. The mid-90s games machine will be more powerful than this. Screen memory-maps for 2000x1500 pixels shouldn't be beyond it – but there'll be no affordable TV or monitor to display that kind of picture.

The problem is that computers are best at producing highly detailed pictures in relatively few colours – 256 is ideal – while TVs are best at producing less detailed pictures in far more colours. The TV approach actually looks better: shapes blur into each other so that, unless you're very close up, you can't actually see the individual pixel boundaries. On a computer display there's no


Solid 3D displays like this really need a "tritter" - a triangle blitter - to draw the basic shapes that the picture's made up of. While the designers are about it, they could knock up a "trigger" chip too, to handle the 3D trigonometry of the game objects.

It's not the resolution of the Konix Multi-System that makes the display so attractive: it's the 256 colours on-screen at once. A typical television display has a 'resolution' of around 320*200 pixels only, but the enormous colour range tricks the eye into seeing a more detailed picture.

blurring however, and diagonal lines have a jagged edge that kills off the picture's realism.

There is a computer technique called "anti-aliasing" (or sometimes "de-jagging") that merges shapes into one another smoothly, giving a TV realism that's extremely attractive. The problem here is that anti-aliasing is extremely difficult to program, and takes way too much CPU time to be used in a normal game. What's needed is a hardware solution, and this is where the video chip comes in. Suppose that your machine has a screen memory map of, say, 1600x1000 pixel-descriptions in 256 colours. Instead of trying to find a monitor that'd diplay 1600x1000 pixels, you could build a smart video chip that turned the memory map into a 320x200 pixel picture for a normal TV – that is, a picture with a fifth of the resolution each way. It would work out the colour of each TV pixel by looking at the corresponding 5x5 area of pixel-descriptions and averaging their values out. (See diagrams for an example of how this would work.)

With a "de-jagging" video chip like that, your processors (CPU, tritter, super-sprite chip etc) could work the way they like to – high-resolution and not too many colours – while your ordinary 625-line colour telly still gets a low-res picture it can display. The de-jagging would go a long way towards smoothing off the edges of the tritter's shapes and – if it's used as the final video process – would also eliminate some of the uglier effects of super-sprite resizing and rotation.

TOWARDS A BETTER DISPLAY

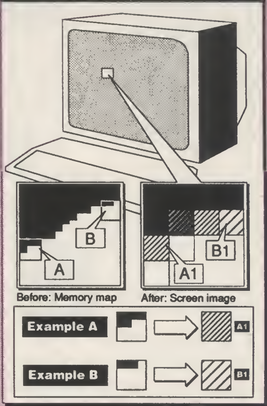
To see how the de-jagger would work, let's look at one tiny section of the 1600x1000 pixel-description memory-map. For simplicity we'll suppose the image is one edge of a black triangle, drawn against a white background.

The memory-map edge is too detailed to be displayed on a TV, but the de-jagger can cope with this. It divides the area we're looking at into 5x5 chunks – there are 16 of them here – and takes an average of the pixel-desciption values in each. It's this average value that's used for the pixel on the finished TV screen display.

Where all 25 descriptions in a chunk are black, the resulting TV pixel will be black; all white, and you'll end up looking at a white pixel on the TV. If there's a mixture of the two, you'll get a grey TV pixel – the shade will depend on the balance of black and white descriptions.

In the case of chunk A, for instance, there are eight black descriptions – 32% of the total – so you'll get a 32% grey value for the corresponding pixel A1. Chunk B has far fewer black descriptions – only 3, or 12% of the total -so the resulting pixel B1 ends up being a 12% grey.

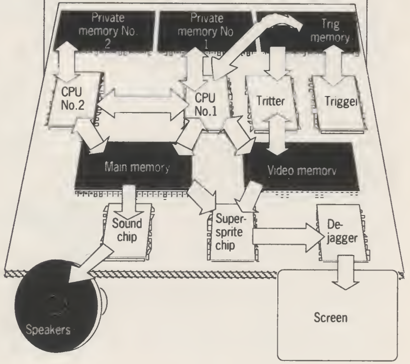
Figure 3 De-jagging at work



THE FUTURE...

Here at ACE we believe that computer entertainment is THE medium of the future. And although we reckon developments such as CDI and Interactive Video (see ACE Issue 26) may eventually rival computers and consoles as games machines, we're sure there'll always be a place for the 'traditional' keyboard/mouse interfaced machine. This booklet has outlined some of the promises, possibilities, and problems – now here's our diagram and 10-point outline specification of the machine of the future, and our challenge to the industry to produce and support it!

Figure 4 Games machine for the 90's



This games machine for the '90s may look complex, but real models might well be even more intricate. Here we have two of everything - two CPUs for games control, two tritters for polygon and image work, and two triggers for 3D co-ordinate maths. The CPUs and the trigger each have their own private pools of high-speed SRAM memory, so that they can work simultaneously without interference.

Once the three sets of processors have created the screen background, the super-sprite chip overlays this with sprite images enlarged (as in ThunderBlade) or rotated from data in the machine's main memory. The resulting picture is then reduced and smoothed by the de-jagger (see Fig 4) ready to be displayed by your TV.

Notice how the number-cruncher has disappeared from this new machine design. With the special-purpose triggers to handle game geometry we can turn the cruncher over entirely to wave-form generation, building it into the sound chip for simplicity.

THE CHALLENGE

There are obviously many factors to take into account here, but we believe that any machine following at least these ten precepts has got to be a winner! And, of course, it could also win its manufacturer – and the programmer(s) of the first ACE-rated 900+ game – £10000! So keep these specs to hand over the next three years so you can tell if and when we'll be forking out our £20,000 – and you should be forking out for a new micro!

THE SPEC

1. **A tritter capable of redrawing the entire screen area twice every 50th of a second.**
2. **A trigger that can rotate, translate, or scale at a rate of 10,000 vertices per frame.**
3. **Two identical CPU's, each 32 bits wide and capable of accessing memory at 10mhz (10 million times a second). This should not be confused with clock speed (which would obviously be faster (see p.6).**
4. **1 million internal pixels and 256 colours on-screen drawn from an 18-bit palette (0.25 million colours).**
5. **A dejagger - must produce a TV-compatible signal and be able to reduce a square grid to a TV pixel.**
6. **16K of private SRAM for each processor.**
7. **At least 1Mbyte of main memory.**
8. **A supersprite chip capable of drawing 1000 sprite images per frame, with each one scaled, rotated, and duplicated as required.**

   And finally, two all important points that some manufacturers never seem to get right:
9. **Development systems must be with software houses at least six months prior to launch, enabling a reasonable software base to be generated for the machine.**
10. **Production levels must be geared to satisfy demand and deliver machines on time!**

Manufacturers who would like to take up the ACE Challenge should get in touch with us for a copy of the complete rules and conditions. The address is The ACE Challenge, 30-32 Farringdon Lane, London, EC1R 3AU. Come on – let's hear from you!